

# Integrating XML and FrameMaker

WHITE PAPER



Sarah O'Keefe  
President

**FrameMaker provides solid support for XML-based authoring workflows. Its PDF and print output capabilities are stellar. Because FrameMaker combines authoring and publishing in a single application, configuring XML support can be quite challenging.**

**It sounds too good to be true: store information in an application-independent, platform-independent format and then render that information through the software of your choice. XML does, in theory, deliver on this promise, but implementation is rarely as straightforward as you might hope. As soon as you choose a publishing tool, your workflow is constrained by that tool's specific limitations, restrictions, and requirements. For example, XML theoretically supports output in any language, including languages that read from right to left (such as Hebrew and Arabic). Most publishing software, however, is more limited; if your publishing tool doesn't support the target language, XML's extensive language support is irrelevant.**

This white paper describes how to integrate XML-based content with Adobe FrameMaker. FrameMaker is an excellent choice for working with XML, but customizations are required to fully support the importing and exporting of XML content.

This white paper assumes basic familiarity with XML and structured authoring; for background information on those topics, refer to the [Structured authoring and XML](#) white paper.

**NOTE:** Because FrameMaker provides custom functionality for the Darwin Information Typing Architecture (DITA), many of the issues described here do not apply to DITA implementation. For more information, see Scriptorium Publishing's [Assessing DITA as a foundation for XML implementation](#) white paper.



## FrameMaker overview

Adobe FrameMaker is a powerful document processor especially suited for long, technical documents, such as reference manuals, user guides, and technical reports. Some feature highlights include:

- ❖ Automatic management of page, chapter, and paragraph numbering (such as figure captions)
- ❖ Flexible table editor
- ❖ Generation of tables of contents and indexes
- ❖ Support for cross-references

FrameMaker produces excellent print and PDF output. When combined with XML, you can use FrameMaker as an XML rendering engine, where you input information as completed XML files and output print and PDF. You can also use FrameMaker as an authoring tool, in which case you create content in FrameMaker and then create XML files. In either case, integrating FrameMaker and XML presents significant technical challenges.

## Structured and unstructured FrameMaker

When you install FrameMaker, you actually install two products—structured FrameMaker and unstructured FrameMaker. Unstructured FrameMaker is the traditional, paragraph-based document processor. Structured FrameMaker was called FrameMaker+SGML and licensed separately in previous versions of FrameMaker. Since version 7, unstructured and structured FrameMaker are covered by a single software license, and you can switch back and forth between the two interfaces by changing a preference setting and restarting FrameMaker (Figure 1).

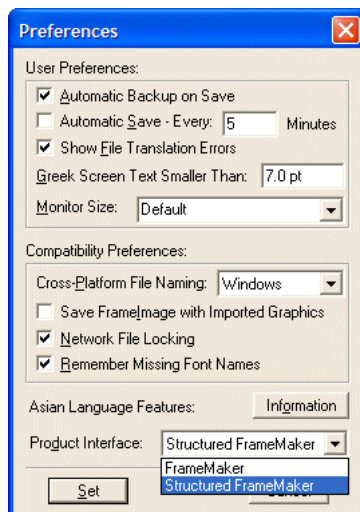


Figure 1: Changing from unstructured to structured FrameMaker

Your project requirements determine whether you need unstructured or structured FrameMaker. You can produce XML files from unstructured FrameMaker, but for most other XML-related tasks, you must use structured FrameMaker, as shown in the following table:

Task	Requires
Opening XML files in FrameMaker	Structured FrameMaker
Exporting XML files from FrameMaker	Unstructured or structured FrameMaker
Importing and exporting (round-tripping) XML	Structured FrameMaker
Validating content against a predefined structure	Structured FrameMaker
Authoring with elements and attributes	Structured FrameMaker
Creating a hierarchical set of elements	Structured FrameMaker

Unstructured FrameMaker allows you to create XML files from your regular, paragraph-based information. You can use the following methods to create XML files:

- ❖ FrameMaker’s built-in Save As XML functionality
- ❖ WebWorks Publisher (Standard or Professional)

In either case, the XML file that’s produced contains a flat sequence of tags with no organization of elements into higher-level elements. The heading and body paragraphs are parallel to each other, and are not grouped into a higher-level element. See (Figure 2).

```

<h2-head-2>Creating a login account</h2-head-2>
<b-body>The login account includes basic information, such as a login
name, and email address. When you create a login account, your password
is emailed to you to verify your email address.</b-body>
<b-body>You can create a maximum of five login accounts on one
computer. When you reach the maximum, <b-bold> Create New
Membership</b-bold> is grayed out in the <b-bold> Login</b-bold> menu.
Contact Scrippsville technical support for assistance.</b-body>
<b-body>You have the option of browsing through Scrippsville without
logging in. This means that you don’t give Scrippsville your name, email
address, and so on; however, if you don’t create a login account and then
log in, you won’t be able to create a Friends list and chat. For this reason,
it’s important to create a login account. Because membership is free, you
don’t lose anything if you decide later not to participate.</b-body>

```

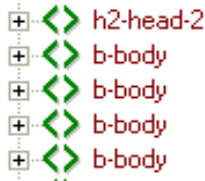


Figure 2: Flat XML produced from unstructured FrameMaker

To create XML with hierarchical, nested elements, as shown in (Figure 3), you need to use structured FrameMaker. (For more information about hierarchy, see Scriptorium Publishing’s [Structured authoring and XML](#) white paper.)



```

<Section>
  <Title>Creating a login account</Title>
  <Para>The login account includes basic information, such as a login
name, and email address. When you create a login account, your password is
emailed to you to verify your email address.</Para>
  <Para>You can create a maximum of five login accounts on one
computer. When you reach the maximum, <MenuItem>Create New
Membership</MenuItem> is grayed out in the <MenuItem>Login</
MenuItem> menu. Contact Scrippysville technical support for assistance.</
Para>
  <Para>You have the option of browsing through Scrippysville without
logging in. This means that you don't give Scrippysville your name, email
address, and so on; however, if you don't create a login account and then
log in, you won't be able to create a Friends list and chat. For this reason, it's
important to create a login account. Because membership is free, you don't
lose anything if you decide later not to participate.</Para>
</Section>

```



Figure 3: Hierarchical elements require structured FrameMaker

It's possible to create attributes in unstructured FrameMaker, but this requires you to set up special constructs, such as conditional text or markers, to embed the attribute information. You would then use special processing in WebWorks Publisher Professional to create the attributes. WebWorks Publisher Standard and the built-in Save As XML conversion cannot support this type of complex transformation.

In structured FrameMaker, however, you can easily assign attribute values to elements and transfer those attributes to XML on conversion (Figure 4).

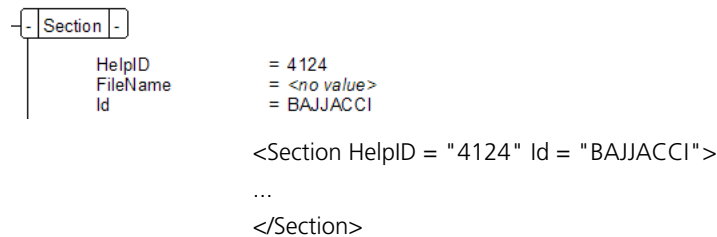


Figure 4: Attribute information in structured FrameMaker and XML

The remainder of this white paper focuses on integration of XML with structured FrameMaker, which enables you to create element-based content. You can automatically assign formatting to elements, and you can transfer information to and from XML.

## Components of a structured FrameMaker solution

Setting up FrameMaker to support XML publishing is a significant effort. At a minimum, you will need the following components:

- ❖ Element definition document (EDD)
- ❖ Document type definition (DTD)

- ❖ Template file
- ❖ Read/write rules files
- ❖ Structured application definition (the listing of configuration files)

The basic FrameMaker/XML integration is shown in Figure 5.

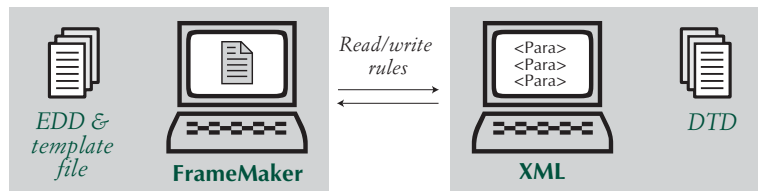


Figure 5: Configuring FrameMaker for XML support

Each of these components is discussed in more detail in the sections that follow.

## Element definition document (EDD) and document type definition (DTD)

The EDD specifies the permitted (or required) structure of the FrameMaker files and provides formatting information for each element (Figure 6).

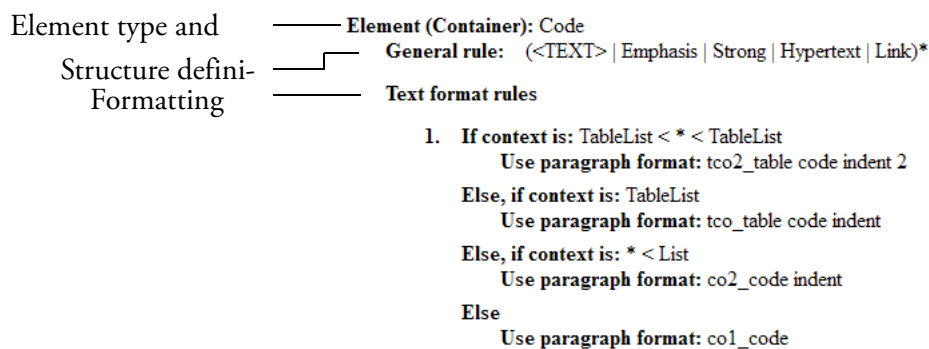


Figure 6: An element definition in an EDD

The document type definition (DTD) provides the equivalent structure definitions for XML:

```
<!ELEMENT Code (#PCDATA | Emphasis | Strong | Hypertext | Link)* >
```

Both EDDs and DTDs contain element definitions, attribute definitions, and comments. The EDD also contains the following additional components:

- ❖ Formatting information—implemented with references to tags in a formatting template, or with information embedded in the EDD itself. Even if you embed information, you will need a formatting template for items such as master page layouts.
- ❖ Element type information—maps elements to FrameMaker objects. For example, index markers must be defined as marker object elements.
- ❖ Miscellaneous settings—provides some additional information, such as the name of the structured application.



FrameMaker allows you to save an EDD as a DTD and vice versa. You can create a complete EDD, export it as a DTD, and have the files you need. If, however, you create the DTD first, you will need to add FrameMaker-specific information (formatting and element types) after you create the EDD from the DTD.

## Template file

When you import an XML document into FrameMaker, you specify that information should flow through a template file. The template file contains structure definitions and formatting information, so putting structured information through the file results in a fully formatted, structured FrameMaker file. The template file must be empty; you cannot put any content in the body pages of the document. The easiest way to create a template file is to import the EDD into the formatting template file and save the result with a new name.

## Read/write rules files

The read/write rules file controls conversion settings for XML import and export. There is an entire read/write rules language that lets you make changes to the default processing of structured information. For example, you can use read/write rules to specify output formats for graphics during conversion (Figure 7).

```

element "AnchoredFrame" {
  is fm graphic element "AnchoredFrame";
  writer facet default {
    convert referenced graphics;
    export to file "${entity}.gif" as "GIF";}
  writer anchored frame {
    export dpi is 150;
    export to file "${entity}.gif" as "GIF";}
  }

```

Figure 7: Excerpt from a read/write rules file

## Structured application

A structured application is a specific implementation of structured authoring (or, an “application of structure”). The term application is often used to mean a specific piece of software, but in structured FrameMaker, an application is a collection of configuration files with a name.

You define the structured application in a special configuration file, `structapps.fm`, which is found in the structure directory inside the FrameMaker installation directory (for example, `C:\Program Files\Adobe\FrameMaker7.1\structure`). Figure 8 shows an excerpt of a `structapps.fm` with a sample application definition.

```

Application Definition Version 7.0
-----
Application name:      DocFrame2
External X-Ref:
  Change Reference To .XML:Enable
DOCTYPE:              UserGuide
                     TitlePage
                     CopyrightPage
                     AboutAuthors
                     TOC
                     LOT
                     LOF
                     Introduction
                     Chapter
                     Glossary
                     Appendix
                     Index
DTD:                  $STRUCTDIR\xml\DocFrame2\docframe.dtd
Template:             $STRUCTDIR\xml\DocFrame2\docframetemplate.fm
Read/write rules:    $STRUCTDIR\xml\DocFrame2\rules.txt

```

Figure 8: Excerpt from a structured application definition file

At a minimum, the structured application must specify the application name, location of the DTD file, the template file (which also contains the structure definitions from the EDD), and the read/write rules file.

FrameMaker ships with a few default structured applications, including an implementation of DocBook, an open standard for technical documentation. A few vendors offer commercial applications, such as the DocFrame implementation from Scriptorium Publishing. If such a solution is a good fit for your content, you may be able to reduce the amount of development time required by building on an existing implementation.

## Starting points

The complexity of the XML-FrameMaker integration depends on your starting point—what format are your current files in and what, if any, requirements have already been defined for the structure? The most common starting points, listed here in order of difficulty, are as follows:

- ❖ Unstructured FrameMaker to structured FrameMaker
- ❖ Word to structured FrameMaker
- ❖ XML-based authoring with FrameMaker as the rendering tool

For a detailed discussion of the high-level implementation process, refer to the Scriptorium Publishing white paper, [Managing implementation of structured authoring](#).



## Unstructured FrameMaker to structured FrameMaker

Moving from unstructured FrameMaker to structured FrameMaker requires you to create an EDD that matches the implied structure in your FrameMaker files. See Figure 9, which shows the structure extracted from a lesson plan.

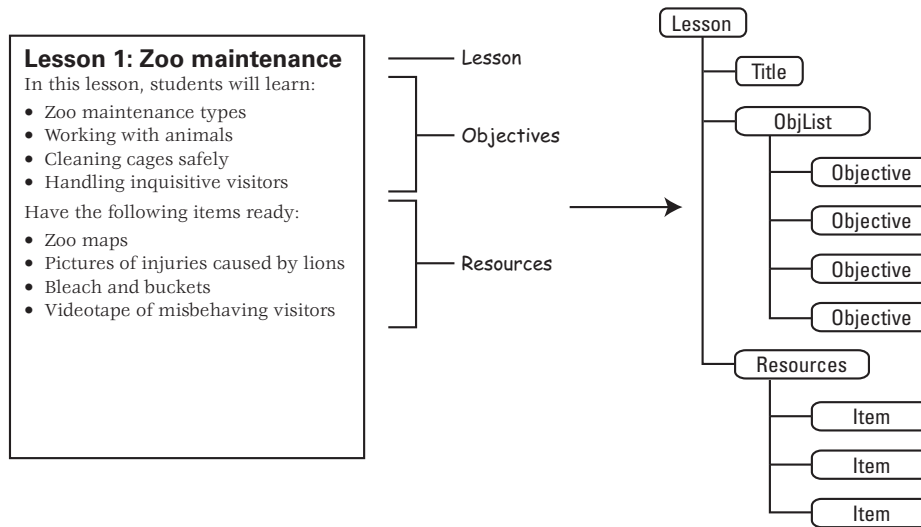


Figure 9: Extrapolating structure from unstructured FrameMaker files

If you have used a formatting template consistently with few or no overrides, conversion of the unstructured files is simplified.

Implementation in FrameMaker requires the following basic steps:

1. Review existing files and extrapolate structure from them.
2. Create an EDD that describes the document structure.
3. Add formatting information to the EDD, either by referencing template components or by embedding formatting in the EDD.
4. Convert unstructured documents to structure.
5. Create scripts to automate cleanup of common conversion problems (deleting extra paragraph tags used as graphic anchors, for example).

Structured FrameMaker is a superset of unstructured FrameMaker, so the document model used in unstructured FrameMaker will map to structured FrameMaker. This simplifies conversion from unstructured FrameMaker. Conversion difficulty is increased by the following factors:

- ❖ Formatting overrides and inconsistent/underdeveloped templates. In structured FrameMaker, you must account for all authoring possibilities ahead of time; authors cannot create elements on the fly. It is difficult to convert files to structure when the source files use a poorly implemented template, have significant overrides, or use different (author-created) tags in each file.
- ❖ Inconsistent or absent document organization.

- ❖ Complex, manual formatting (for example, a single table that's been modified with custom ruling and shading to look like two tables with a gutter in between them).
- ❖ Metadata required in the structured files for which there is no equivalent in the unstructured files. A common example is that generic sections (using a Heading2 tag for the title) become several different types of sections in the structure. It may be impossible to map a single paragraph tag to a Reference, Procedure, or Concept element depending on the type of information in the section.
- ❖ Documents with multiple flows. In structured FrameMaker, each flow has its own structure. You cannot associate information in different flows. For example, if you set up an instructor guide with a Student flow and an Instructor flow side by side, the two flows do not have any relationship, except that they are positioned on each page to look related.
- ❖ Formatting “cheats” where the document's visual appearance is different from the underlying components. For example, tables are often used for nontabular data, such as bulleted items in multiple columns or one table with custom ruling and shading to mimic two side-by-side tables. These work-arounds cause problems when you attempt to process the structure later.

If your source files are in unstructured FrameMaker, you need not worry about product-specific issues such as language support (any language you're using in unstructured FrameMaker is also supported in structured FrameMaker).

Conversion from any unstructured word processor format to a structured environment is challenging, but the close relationship between unstructured and structured FrameMaker make that conversion slightly easier.

## Word to structured FrameMaker

Like unstructured FrameMaker, Microsoft Word is a word processor. Converting Word files to structured FrameMaker requires the same basic steps as conversion of unstructured FrameMaker files. There are, however, some additional complications:

- ❖ You cannot use a Word template as a basis for formatting in structured FrameMaker, so you need to create a FrameMaker formatting template.
- ❖ Conversion from Word into structured FrameMaker is more complex than conversion from unstructured FrameMaker. You can use scripts for pre- or post-processing files during conversion to reduce manual cleanup on the files.
- ❖ Word's table model is quite different from FrameMaker's. For example, Word permits nested tables and tables that do not follow a regular grid; FrameMaker does not.
- ❖ Word macros do not convert into FrameMaker.

Word files with a template and consistent styles are much easier to convert than files that use the Normal style with ad hoc formatting.



## Predefined XML structure to structured FrameMaker

If you are required to adapt structured FrameMaker to use an existing XML DTD or schema, you can expect to have significant implementation challenges. Structured FrameMaker, like all publishing tools, has a specific document model. If the document model described in the DTD does not match the FrameMaker document model, you may need to do considerable programming work to make the two models compatible. The FrameMaker document model is described in the section that follows.

## The FrameMaker document model

Like any publishing tool, FrameMaker has layout limitations. If your structure does not match the default FrameMaker document model, you may be able to customize FrameMaker (see page 16), but expect implementation costs to skyrocket.

### Tables

FrameMaker has a very powerful, flexible table editing feature. XML, however, has no limitations whatsoever on the table structures. Some of the FrameMaker requirements include the following:

- ❖ A table cannot occur inside another table.
- ❖ Each row in a table must have the same number of columns. You cannot create an irregular table, except by straddling cells.
- ❖ Tables are always defined row by row, never column by column.
- ❖ Each table can have only one table body section.

Out of the box, FrameMaker supports the CALS table model, a standard for structured tables. If your tables are set up using CALS structure, you should be able to process them successfully, subject to the limitations described here. For more details about how FrameMaker handles CALS tables by default, refer to Appendix A of the *Structure Applications Developer's Guide*, an online manual that is installed with FrameMaker.

### Object elements

XML does not make any distinction between block elements (paragraphs), inline elements (text ranges), graphic elements, and so on. In FrameMaker, your structure definitions must specify the type of element being created. For example, you must explicitly specify that the CrossRef element is a cross-reference object. See Figure 10.

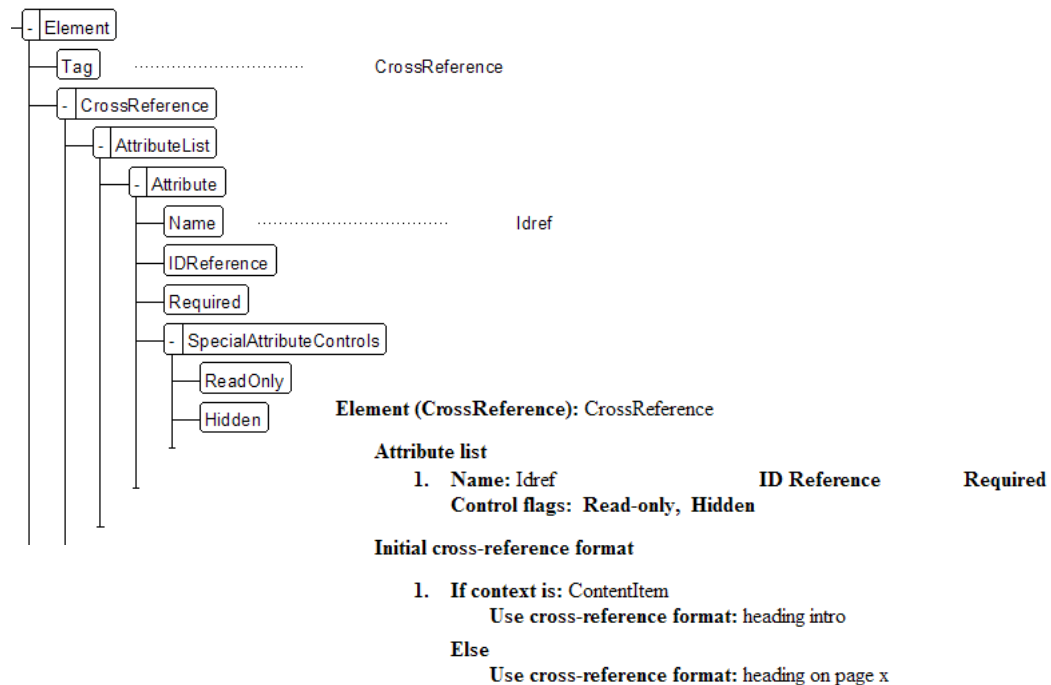


Figure 10: Object elements require special definitions in the EDD

Most elements are container elements. A container element is an element that allows text or additional elements inside the element. Container elements are used for paragraphs, text ranges, and grouping elements (which may not contain text but do have child elements). For tables, FrameMaker provides additional specialized container element types.

The elements that require special handling are:

- ❖ System variables
- ❖ Cross-references
- ❖ Table components: table, table heading, table body, table footing, table row, table cell, and table title
- ❖ Rubi groups (used mainly in Japanese publishing)
- ❖ Markers
- ❖ Graphics
- ❖ Equations

FrameMaker defines a special class of elements, called object elements, which are not permitted to have child elements or contain text. System variables, cross-references, markers, graphics, and equations are all object elements. Object elements can have associated attributes, but the elements cannot contain child elements or text.



You can work around these limitations by customizing your structured application files, but they do increase the complexity of the implementation. For instance, FrameMaker expects the following structure for an index marker by default:

```
<IndexEntry text="index entry text"/>
```

You might, however, have XML where the IndexEntry element has text:

```
<IndexEntry>index entry text</IndexEntry>
```

You would need to modify the import process to move the index information from the element text to the attribute text. Furthermore, there is a 255-character limit on marker text in FrameMaker.

The syntax used for index entries with multiple levels may also present challenges. FrameMaker embeds the entire index marker text in a single attribute. Primary and secondary entries are separated with colons; multiple entries use colons, as shown here:

```
<IndexEntry text="bear:hibernating;hibernation"/>
```

If the XML structure uses different indexing syntax, you will need to map it over to conform to FrameMaker's expectations.

For all object elements, keep in mind that you cannot have child elements inside the object.

## Generating unique IDs for elements

ID attributes are commonly used to manage cross-references in structured documents, and FrameMaker supports these IDs. When you insert a cross-reference element and point to another element, FrameMaker automatically creates an Id attribute with a unique value for the target of the reference. The cross-reference element automatically gets an Idref attribute with the same value (Figure 11).

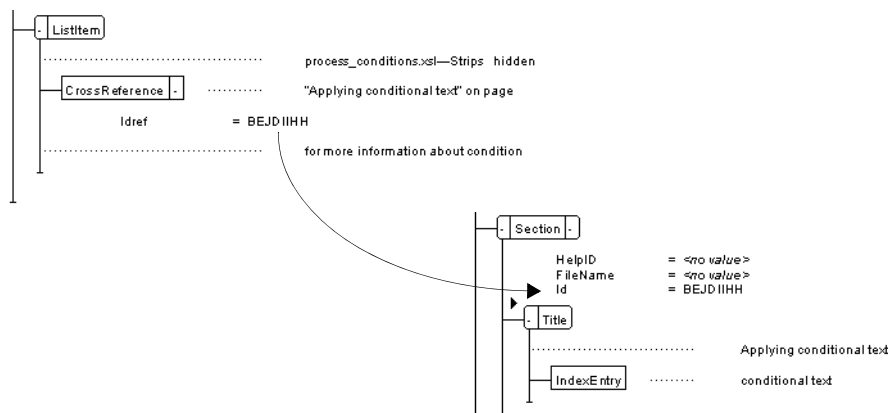


Figure 11: Structured cross-references

To implement structured cross-references in FrameMaker, you must set up your structured definitions as follows:

- ❖ Require an Idref attribute for every element of type cross-reference (Figure 10).
- ❖ Permit an Id attribute for any element that could become the target of a cross-reference (Figure 12).

FrameMaker will not automatically assign Id values to elements as you create the elements; Id values are assigned only when the specified element becomes the target of a cross-reference. Once an Id value is assigned, it is reused for any subsequent cross-references.

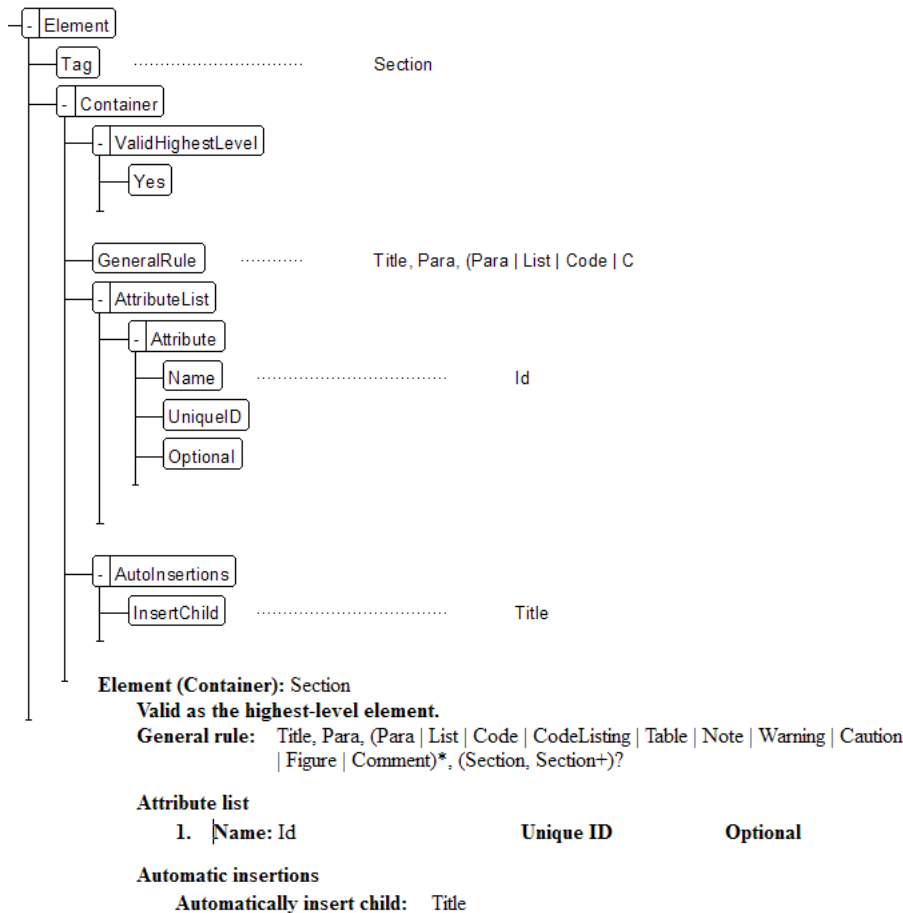


Figure 12: Structure definitions for target element

## Generated files

Generated files, such as the table of contents and index, are unstructured in FrameMaker, even if the other files in the book are structured.

You can set the read/write rules to drop the unstructured generated files on export to XML. If you are processing the XML to produce rendered output (such as HTML), you can re-create the table of contents and index from the information present in the XML files (that is, the various title elements for the table of contents and the structured index entry elements for the index).

You can set the read/write rules in the structured application to create a book file and chapter files during importing of XML content. Once the content is in FrameMaker, you create new generated files for the book. You can also write a FrameMaker Developer's Kit (FDK) client to automatically create the generated files during the import process.



## Labeling versioned information with conditional text and attributes

FrameMaker’s conditional text feature gives you the ability to show and hide information based on the condition settings.

In version 7.0 and earlier, shown conditional text in the FrameMaker files is exported to XML, and hidden conditional text is deleted. The resulting XML files do not contain any information about the conditional text labels.

In version 7.1, you can successfully round-trip conditional text information with processing instructions. Figure 13 shows a recipe with two conditions and how that conditional information is maintained with processing instructions in the XML output.

### Cream Cheese Icing

	8 ounces	<u>light</u> cream cheese, <i>softened</i>	reduced_fat condition
	1/2 cup	butter or margarine, <i>softened</i>	
	4 cups	confectioners sugar	
	1 tsp.	vanilla extract	
no_nuts condition	-1 cup	<del>pecans, chopped</del>	

Combine the cream cheese and butter in a mixing bowl. Using a hand mixer, beat the cream cheese and butter until fluffy.  
Gradually blend in the confectioners sugar, then blend in the vanilla extract.  
~~Stir in the pecans.~~

```

<?Fm Condition no_nuts Red STRIKETHROUGH show?>
<?Fm Condition reduced_fat Blue SINGLE_UNDERLINE show?>
<Recipe>
<Name>Cream Cheese Icing</Name>
<IngredientList>
<Ingredient><Quantity>8 ounces</Quantity><Item>
<?Fm Condstart reduced_fat?>light <?Fm Condend reduced_fat?>cream
cheese</Item><PrepMethod>softened</PrepMethod></Ingredient>
<Ingredient><Quantity>1/2 cup</Quantity><Item>butter or margarine
</Item><PrepMethod>softened</PrepMethod></Ingredient>
<Ingredient><Quantity>4 cups</Quantity><Item>confectioners sugar
  
```

Figure 13: Conditional text processing in a recipe (processing instructions and conditional content in bold type)

We recommend that you avoid “double-tagging” text with more than one condition if possible; it can make processing the XML very complicated.

Another option for versioning information is to use element attributes. Attributes are better integrated into the structure of the document (Figure 14).

```

<Chapter>
<Para output="print">This book represents your first step in the Scripppyville
world.</Para>
<Para>Scripppyville provides a new way to experience the Internet...</Para>
...
</Chapter>

```

Figure 14: Using attributes for conditional information

Attributes are easier to process in XML than processing instructions. FrameMaker 8 includes support for attributes, but FrameMaker 7 does not have the functionality to show and hide information based on attribute values. If you use attributes to label information as belonging to different versions, you will need to develop a solution for displaying and printing the different versions of the document. Here are some options:

- ❖ If information is being created in XML, preprocess the XML files with an Extensible Stylesheet Language (XSL) script that strips content based on attribute values. For example, if you have an Output attribute, you could remove any material that has the OnlineOnly attribute value before you import the content into FrameMaker for printing.
- ❖ Use a third-party plug-in for FrameMaker called Sourcerer, which allows you to show and hide information based on attribute values.
- ❖ Use FrameScript or the FDK to enable attribute-based versioning.
- ❖ Export FrameMaker files to XML, process them to eliminate information you don't want, and re-import the processed file for printing.

## Element sequence

You can process an XML source document and produce rendered documents (perhaps in HTML) in which information is presented in a different sequence from the original XML file.

FrameMaker can interpret and format element information, but as a general rule, the elements need to be in the same order as the page-based presentation. If the element sequence in your source XML files does not match the required sequence, preprocess the XML file to create a file with the needed element order and then import the processed file for printing.

## Formatting issues

In the FrameMaker EDD, you use context rules to define formatting for elements. The context rules let you specify that a single element uses different formatting based on its position (context) in the structure (Figure 15).



**Element (Container):** Title  
**General rule:** (<TEXT> | IndexEntry)\*

**Attribute list**

1. <b>Name:</b> Id	<b>Unique ID</b>	<b>Optional</b>
<b>Control flags:</b> Read-only, Hidden		

**Text format rules**

1. **If context is:** {first} < Chapter  
**Context label:** Chapter  
**Use paragraph format:** h1\_head 1  
**Else, if context is:** \* < CodeListing  
**Context label:** CodeListing  
**Use paragraph format:** cot\_code title  
**Else, if context is:** Section < Section < Section < Section  
**Context label:** Head5  
**Use paragraph format:** hr\_heading runin  
**Else, if context is:** Section < Section < Section  
**Context label:** Head4  
**Use paragraph format:** h4\_head 4  
**Else, if context is:** Section < Section  
**Context label:** Head3  
**Use paragraph format:** h3\_head 3  
**Else, if context is:** Section  
**Context label:** Head2  
**Use paragraph format:** h2\_head 2

Figure 15: Context rule example

Context rule syntax lets you check for a particular parent, ancestor, or ancestor sequence and apply formatting based on the match. You can also use sibling rules; for example, a rule applies formatting based on whether the current element is the first sibling. Context rules are more limited than what's available in the XSL used for processing XML. Context rules do not let you assign formatting based on descendant elements. The sibling rules are also more limited than in XSL.

## FrameMaker customization

In addition to FrameMaker's built-in features, there are several ways to customize default behavior. These range from inexpensive plug-ins to API programming. The FDK enables programmers to make extensive changes to default behavior and to write custom code to work around application limitations. Although the FDK software is free, FDK programming projects are expensive. Where possible, we recommend accommodating FrameMaker's default behavior to reduce configuration costs.

Two scripting languages, FrameScript and FrameAC, provide easier access to FDK functions. The FDK supports deeper, more complex customizations, but for many requirements, FrameScript or FrameAC provides an inexpensive alternative to FDK programming.

## Language support

XML uses the Unicode character set. The goal of Unicode is to provide a single character set encoding that covers all the languages in the world. This includes the Latin alphabet (English and most Western European languages), Cyrillic alphabet (Russian and other Eastern European languages), character-based languages (such as Japanese, Chinese, and Korean), and languages that read from right to left (Hebrew and Arabic).

FrameMaker 8 now provides Unicode support. Although, right-to-left languages (Arabic, Hebrew) and complex scripts (Thai, Vietnamese) are not supported. FrameMaker supports the following language groups:

- ❖ English and Western European languages (French, German, Italian, Spanish, Norwegian, and so on)
- ❖ Double-byte languages (Japanese, Korean, simplified Chinese, and traditional Chinese)

FrameMaker supplies dictionaries for about two dozen languages; for a list, refer to the Adobe web site. In addition to the officially supported languages, it's possible to configure FrameMaker to process additional languages, such as Russian, Polish, Greek, and Turkish. FrameMaker will not render any bidirectional or right-to-left languages, such as Arabic or Hebrew.

Adobe has acknowledged that full Unicode support is a significant issue for FrameMaker. However, the company has not specified whether or when such support might be implemented.

## Entities

In XML, an entity is a placeholder. Entities can be placeholders for text; system entities are references to external files. Entities are most often used for three different types of information:

- ❖ Graphics
- ❖ XML fragments
- ❖ Text fragments (such as copyright statements or other often-repeated text)

When working with FrameMaker, each of these entity types is handled differently.

### Graphics

FrameMaker uses an object element for referenced graphics. When you export to XML, the graphic element contains a reference to an entity name in an attribute, as shown in the following example (attributes other than entity have been truncated, as indicated by the ellipsis):

```
<ImportedGraphic entity = "ImportedGraphic1" .../>
```

At the top of the XML output file, you will find a definition for the entity:

```
<!ENTITY ImportedGraphic1 SYSTEM "ImportedGraphic14.gif" NDATA gif>
```

This type of entity is an unparsed entity reference. The entity is unparsed because when the content is displayed as XML, the entity name is not replaced with the entity definition itself.

### XML fragments

You can use entities to refer to external files that are parsed. In this case, you reference the entity in your main XML flow, as shown here:

```
<Book>
  &ch1;
</Book>
```



At the top of the XML file, you will find an entity definition for ch1:

```
<!ENTITY ch1 SYSTEM "file01.xml">
```

When you export a structured book from FrameMaker to XML, the default handling is to create a system entity for each file in the book.

On import from XML, entities are not used to generate book component files. Instead, you must specify which elements will start new files in the read/write rules:

```
put element "Chapter" in file "ch.fm";
put element "Appendix" in file "app.fm";
put element "Glossary" in file "gloss.fm";
```

You can also use processing instructions to specify where to begin a new file and what file name to use.

## Building document structures

Inside FrameMaker, the EDD controls document structure, just as a DTD or schema controls structure in XML files. There are three ways to create the EDD:

- ❖ If you are using a schema, you can convert the schema to a DTD, which can then be imported into FrameMaker to create an EDD
- ❖ If you have an existing DTD, you can import the DTD into FrameMaker to create an EDD.
- ❖ You can create the EDD inside FrameMaker.

Conversion of schema files into EDDs is not supported in FrameMaker, so if you have a schema file, you will need to convert it to a DTD and then import the DTD into FrameMaker.

Schema files provide much more extensive datatyping than DTDs or EDDs do. For example, you can specify that a Year field must contain only four digits. This close control over content is not possible with DTD or EDD files.

## Importing a DTD

If you have a DTD already defined for your document structures, you can open the DTD in FrameMaker and convert it to an EDD. This process usually works fairly well, but there are some limitations:

- ❖ FrameMaker does not have a feature equivalent to parameter entities, which allow reuse of structure definitions and attribute lists in the DTD. In the EDD, parameter entities are converted to plain text.
- ❖ If the DTD uses characters not supported in FrameMaker, conversion errors will occur.

## Creating structure definitions in FrameMaker

You can create structure definitions in a FrameMaker EDD and then export this EDD as a DTD. FrameMaker permits several constructs in EDDs that are legal in SGML DTDs but not in XML DTDs. The following should not be used in an XML workflow:

- ❖ Inclusions
- ❖ Exclusions
- ❖ Ampersand connector (&)
- ❖ Structure definitions that require text and other elements, such as:

<TEXT>,Emphasis

For XML, you must substitute the following:

(<TEXT>|Emphasis)\*

If you create your structure definitions in FrameMaker, you must keep the XML limitations in mind to ensure that the EDD will export to an XML DTD without errors.

## Resources and references

For additional information, try one of these resources:

- ❖ *Structure Application Developer's Guide* (in the OnlineManuals folder of the FrameMaker installation directory) provided by Adobe
- ❖ [Structured authoring and XML](#) white paper.
- ❖ [Managing implementation of structured authoring](#) white paper.
- ❖ DocFrame information, [www.docframe.com](http://www.docframe.com)
- ❖ FrameMaker Developer's Kit (FDK), [partners.adobe.com/asn/frame maker/fdk.jsp](http://partners.adobe.com/asn/frame maker/fdk.jsp)
- ❖ FrameScript, [www.framescript.com](http://www.framescript.com)
- ❖ FrameAC, [www.4adobe.com/Server\\_Products/Development\\_Applications/FrameAC/index.jsp](http://www.4adobe.com/Server_Products/Development_Applications/FrameAC/index.jsp)
- ❖ Sourcerer, [www.advantica.biz/sourcerer](http://www.advantica.biz/sourcerer)
- ❖ CALS table model information, [www.oasis-open.org/specs/a502.htm](http://www.oasis-open.org/specs/a502.htm)
- ❖ *XSLT Programmer's Reference* (ISBN 1861005067), Michael Kay
- ❖ Unicode Consortium, [www.unicode.org](http://www.unicode.org)
- ❖ DocBook, [www.docbook.org](http://www.docbook.org)
- ❖ framers list, [www.frameusers.com](http://www.frameusers.com)
- ❖ FrameSGML list, [groups.yahoo.com/group/FrameSGML/](http://groups.yahoo.com/group/FrameSGML/)

Scriptorium Publishing Services, Inc.  
PO Box 12761  
Research Triangle Park, NC 27709-2761  
USA  
info@scriptorium.com  
919-481-2701  
[www.scriptorium.com](http://www.scriptorium.com)

